INFORMATION AND CONTROL 33, 253–272 (1977)

Grammar Enumeration and Inference

R. M. WHARTON

Department of Computer Science, York University, Downsview, Ontario, Canada M3J 1P3

An unidentified language can be identified by inferring a grammar which generates it. One approach to the problem of grammar inference requires the exhaustive enumeration of the solution space of grammars. While enumerative procedures for grammar inference have important formal strengths, they typically require unacceptably large amounts of computation. A number of techniques are introduced here which significantly improve the efficiency of enumerative inference procedures by eliminating from the enumeration the vast majority of unacceptable grammars. These techniques are implemented in a general purpose grammar enumerator which is readily adaptable to a variety of classes of grammars, criteria for halting, and measures of grammar complexity. Empirical evidence of the efficacy of these techniques is also provided.

1. INTRODUCTION

A language, conceived of as a set of strings over a finite alphabet, may be identified by specifying a grammar which generates it. The problem of identifying a language given finite samples of strings from the language is known as *grammar inference* and is a specific form of the more general problem of *inductive inference.* The significance of the grammar inference problem has been stated previously in Solomonoff (1964), Gold (1967), Horning (1969), and Fu (1974), and will not be reiterated here. An overview of the approaches which have been taken to this problem and of previous results can be found in Biermann and Feldman (1972), Gold (1973), and Fu (1974).

It is convenient to classify grammar inference procedures into two categories: *constructive* procedures and *enumerative* procedures. While this distinction is useful, it should not be considered to be rigorous, since some procedures classed as constructive contain enumerative features, and some procedures classed as enumerative contain constructive features. The chief characteristic of constructive grammar inference procedures is the systematic use of sample strings to construct the rules of the grammar. Various constructive inference procedures have been presented in Solomonoff (1959), Feldman (1967), Feldman *et al.* (1969), Pao (1969), and Crespi-Reghizzi (1970). Enumerative grammar inference procedures exploit the fact that the class of all grammars and most of its more

ISSN 0019-9958

important subclasses can be effectively enumerated. Typically, enumerative methods for grammar inference require enumerating the class of grammars under consideration, examining each grammar in turn, and selecting the first grammar which is appropriate for the sample of strings from the unidentified language. Enumerative inference procedures have been described in Gold (1967), Feldman *et al.* (1969), Feldman (1972), Horning (1969), and Wharton (1974).

By comparing the results obtained from constructive methods with those obtained from enumerative methods, we can see two general advantages of the enumerative techniques. First, enumerative methods are exhaustive, or *complete*; that is, by examining each grammar in sequence, they assure that no relevant grammar is missed. Therefore if the class of grammars being enumerated is ordered by increasing difficulty, or *complexity*, they typically select the least complex suitable grammar. With constructive inference techniques the question arises as to how the grammars that are produced compare with those that are not produced. Typically, it is not claimed that constructive inference procedures produce optimal solutions. Second, with enumerative methods the classes of grammars which may be inferred can be quite large. In Gold (1967), Feldman *et al.* (1969), Feldman (1972), and Wharton (1974) a class of grammars must only satisfy the requirements that it be effectively enumerable and that each grammar be decidable. These restrictions are sufficiently generous that they admit classes of grammars even larger than the class of context sensitive grammars (Hopcroft and Ullman, 1969). On the other hand, whenever constructive methods have been shown to work for complete classes of grammars, the classes are distinctly smaller than the class of context free grammars. For example, Pao (1969) infers *regular grammars* and *delimited grammars*, Gips (in Feldman *et al.*, 1969) infers *pivot grammars*, and Crespi-Reghizzi (1970) infers *free operator precedence grammars*.

Constructive inference techniques have one important advantage over enumerative techniques. The constructive methods frequently require only modest amounts of computation, while the enumerative methods typically require unacceptably large amounts of computation, a direct consequence of the exhaustive search characteristic of this approach. The primary motivation for the work described here is the desire to devise inference procedures that retain the theoretical advantages of enumerative procedures, yet are efficient enough to be applicable to realistic inference problems.

2. Preliminary Definitions and Notation

A *vocabulary* is a finite set of symbols. A *string* is a finite sequence of symbols from a vocabulary. In particular, λ denotes the *empty string*. For any vocabulary

V the *set of strings over* V, denoted V^*, is the *free monoid* generated by the symbols of V under the operation of *concatenation* with λ as the identity element. The *set of nonempty strings over* V, denoted V^+, is defined as $V^* - \lambda$. The *length* of a string α is denoted $l(\alpha)$. For any vocabulary V a *language over* V is a subset of V^*. For convenience we restrict our attention to subsets of V^+, or *λ-free languages*.

A *grammar* G is a 4-tuple (N, T, P, X), where N, T, P, and X designate, respectively, *the nonterminal vocabulary* (*nonterminals*), *terminal vocabulary* (*terminals*), the set of *productions*, and the *initial nonterminal*. N and T are finite, nonempty, and disjoint. $V = N \cup T$ is the *vocabulary* of G. $X \in N$. P is a finite, nonempty set of expressions of the form $\alpha \to \beta$, where $\alpha \in V^+$ and $\beta \in V^*$. If $\alpha \to \beta$ is a production of P and γ and δ are any strings in V^*, then the production $\alpha \to \beta$ may be *applied* to the string $\gamma\alpha\delta$ to obtain $\gamma\beta\delta$. This process is denoted $\gamma\alpha\delta \Rightarrow \gamma\beta\delta$. The *reflexive transitive closure of* $\Rightarrow$ is denoted $\Rightarrow^*$. If at least one production has been applied, we use the notation $\Rightarrow^+$. For any grammar G the *language $L(G)$ generated by G* is defined by $L(G) = \{w \mid X \Rightarrow^* w \text{ and } w \in T^*\}$.

Γ denotes a class of grammars and G denotes a particular grammar in Γ. For any grammar $G = (N, T, P, X)$ we use the symbols X, Y, Z, and W to denote elements of N, with X always denoting the initial nonterminal, and the symbols a, b, c, d, e, $+$, $-$, (, and) to denote elements of T. With these conventions a grammar can be completely specified by listing its productions. If a grammar contains k productions with the same left-hand side, $\alpha \to \beta_1$, $\alpha \to \beta_2$,..., $\alpha \to \beta_k$, where $k \geqslant 2$, then they are denoted $\alpha \to \beta_1 \mid \beta_2 \mid \cdots \mid \beta_k$. For any grammar $G = (N, T, P, X)$ the symbols n, t, and p denote the *cardinalities* of N, T, and P, respectively.

A grammar is *context sensitive* if for every production $\alpha \to \beta$, $l(\alpha) \leqslant l(\beta)$. A grammar is *context free* if every production has the form $Y \to \alpha$, where $Y \in N$ and $\alpha \in V^+$. A grammar is *regular* if every production has the form $Y \to a$ or $Y \to aZ$, where $a \in T$ and $Y, Z \in N$. Any language generated by a context sensitive (context free, regular) grammar is a *context sensitive* (*context free*, *regular*) *language*. The containment relationships among these classes of grammars and languages are described in Hopcroft and Ullman (1969). A context free grammar is in *Chomsky normal form* if the right-hand side of each production has the form a or YZ, where $a \in T$ and $Y, Z \in N$. A context free grammar is in *Greibach normal form* if the right-hand side of each production has the form $a\alpha$, where $a \in T$ and $\alpha \in N^*$. A context free grammar is a *restricted operator grammar* if the right-hand side of each production has one of the forms a, Y, aY, Ya, aYb, and YaZ, where $a, b \in T$ and $Y, Z \in N$. It is well known that any context free language can be generated by a grammar in Chomsky normal form and by a grammar in Greibach normal form (Hopcroft and Ullman, 1969), and it can similarly be shown that any context free language can be generated by a restricted operator grammar. The grammar for arithmetic expressions in Example 1 is a restricted operator grammar.

EXAMPLE 1.

$$G: X \to Y \mid +Y \mid -Y \mid X + Y \mid X - Y,$$
$$Y \to Z \mid Y^*Z \mid Y/Z,$$
$$Z \to a \mid (X).$$

Two grammars are *completely equivalent* if one can be transformed into the other by a one-to-one, onto mapping of their nonterminal vocabularies. The set of all grammars completely equivalent to any one grammar constitutes an *equivalence class*. In the sequel we let each complete equivalence class be represented by any one of its members. The grammars in Example 2 are completely equivalent.

EXAMPLE 2.

$$G_1 : X \to YZ, \quad Y \to a, \quad Z \to b; \qquad G_2 : X \to ZY, \quad Y \to b, \quad Z \to a.$$

Two grammars are *weakly equivalent* if they generate the same language. Clearly, any two completely equivalent grammars are also weakly equivalent, but the converse of this is not generally true. The grammars in Example 3 are weakly equivalent, since $L(G_1) = L(G_2) = \{a, b\}^+$, but are not completely equivalent.

EXAMPLE 3.

$$G_1 : X \to a \mid b \mid aX \mid bX; \qquad G_2 : X \to Y \mid Z, \quad Y \to W \mid WWY, \quad Z \to WW \mid WWZ, \quad W \to a \mid b.$$

A (*static*) *complexity measure*, or *size measure* (Blum, 1967) on a class Γ of grammars with a terminal vocabulary T is a mapping σ, from Γ into the nonnegative integers, which satisfies the following axioms: (1) There exists at most a finite number of complete equivalence classes of grammars of any given complexity. (2) There exists an effective procedure which determines for any c which grammars are of complexity c. These two axioms for complexity are so broadly inclusive that one can define complexity measures which effectively negate all our efforts to enhance efficiency. We therefore define two particular complexity measures which are intuitively reasonable and empirically convenient. The complexity measures are functions of n (the cardinality of N), p (the cardinality of P), m (the length of the longest right-hand side of a production in P), and k (the length of the longest left-hand side of a production in P). Since in the sequel we are only concerned with context free grammars, $k = 1$ and can be

omitted from these functions. In some classes of grammars, such as regular grammars, Chomsky normal form grammars, and restricted operator grammars, m is bounded and therefore can also be omitted from functions defining complexity measures. For any class of grammars in which m is bounded, a simple complexity measure is provided by the following *pairing function* in p and n: $c = ((p + n - 1)(p + n - 2)/2) + n$. When m is unbounded, as it is in the class of Greibach normal form grammars and in the class of context free grammars, we apply the same pairing function again to obtain the complexity measure $c' = ((c + m - 1)(c + m - 2)/2) + m$. The two complexity measures specified here are arbitrary, but they provide adequate examples and are retained throughout the sequel.

For any grammar G let the nonterminal vocabulary N be given an arbitrary order, except that the initial nonterminal must be the first nonterminal. Then G has a *structure*, which we denote by a vector S of length n consisting of nonnegative integers in which S_i is the number of productions having the ith nonterminal on the left-hand side. It is clear from this definition that $\sum_{i=1}^{n} S_i = p$. The regular grammar in Example 4 has complexity $c = 8$ and structure $S = \langle 2, 1 \rangle$.

EXAMPLE 4.

$$G: X \to a \mid bY,$$
$$Y \to cX.$$

Let Γ be any class of grammars over a terminal vocabulary T and let Γ_c be the subset of Γ whose grammars have complexity c, where $c \geqslant 1$. Then Γ is enumerated in order of increasing complexity by enumerating each Γ_c as c takes on successively higher positive values. It is clear from the first axiom for complexity measures that each Γ_c is finite. For each complexity class Γ_c, S can take on a finite number of values, determined only by p and n (and sometimes m). The set of all grammars in Γ_c with the same value of S constitutes a *structure class*, denoted $\Gamma_{c,S}$. A complexity class is enumerated by enumerating in succession each of its structure classes.

Before proceeding to a detailed description of the process of grammar enumeration, we pause to consider the question of when the enumeration process should terminate. In general, enumeration terminates when one or more grammars are found which are *compatible* with the available information about the unidentified language. However, the concept of compatibility can be defined in a variety of ways. For any unidentified language U, a *positive sample* S^+ is a finite set of strings in U and a *negative sample* S^- is a finite set of strings which are not in U. That is $S^+ \subset U$ and $S^- \subset \overline{U}$. We give below three possible criteria for the compatibility of a grammar with a sample. (1) Given a positive sample S^+, a grammar G is compatible with S^+ if $S^+ \subset L(G)$. (2) Given a positive sample S^+ and a negative sample S^-, a grammar G is compatible with the combined sample

if $S^+ \subset L(G)$ and $S^- \subset \overline{L(G)}$. (3) Given a positive sample S^+ whose longest string has length k, a grammar G is compatible with S^+ if the subset of $L(G)$ whose strings do not exceed length k is identical to S^+. It is clear that these three criteria are in increasing order of "strength." Independent of what S^- may be, any grammar satisfying criterion 2 satisfies criterion 1, and if S^- contains no strings longer than k, any grammar satisfying criterion 3 satisfies criterion 2. The third criterion is equivalent to the requirement that the sample be complete up to length k.

3. A General Purpose Grammar Enumerator

We now describe a simple procedure which can be used to enumerate various classes of grammars, including context free grammars, Chomsky normal form grammars, Greibach normal form grammars, restricted operator grammars, and many others. A class of grammars Γ is enumerated by enumerating its complexity classes and a complexity class Γ_c is enumerated by enumerating its structure classes. For example, let Γ be the class of regular grammars, let $p = 3$, $n = 2$, and $c = 8$. Then S takes on the successive values $\langle 3, 0\rangle$, $\langle 2, 1\rangle$, $\langle 1, 2\rangle$, and $\langle 0, 3\rangle$. The grammars in Example 5 illustrate the four structure classes.

Example 5.

$$G_1 : X \to a \mid bX \mid cY; \qquad G_2 : X \to a \mid bY, \quad Y \to cX;$$

$$G_3 : X \to aY, \quad Y \to b \mid cX; \qquad G_4 : Y \to a \mid bX \mid cY.$$

Each structure class $\Gamma_{c,S}$ is enumerated by the *grammar-generating schema* described below. First, the vocabulary V is ordered. We assume that the terminal vocabulary T is known, and we give it a fixed arbitrary order. Once the complexity c is specified, the number of nonterminal symbols n can be determined. The nonterminal vocabulary N is represented by n arbitrary symbols, which are given a fixed arbitrary order, except that the initial nonterminal must be first. The vocabulary $V = T \cup N$ is ordered by placing T before N.

Many of the common subsets of context free grammars are distinguished by restrictions that are placed on the form of the right-hand sides of the productions. For each class of grammars for which m is bounded, once the sequence V is determined, an ordered sequence R of *all possible right-hand sides* can be constructed. The sequence R is ordered by the following rules: (1) For any strings $x, y \in R$, if $l(x) < l(y)$ then x precedes y in R. (2) For any strings $x = a_1 a_2 \cdots a_k$ and $y = b_1 b_2 \cdots b_k$, where $x, y \in R$ and $l(x) = l(y)$, let $a_i = b_i$ for $i = 1, 2, \ldots, j$

and $a_{j+1} \neq b_{j+1}$, where $0 \leqslant j < k$. Then x precedes y in R if a_{j+1} precedes b_{j+1} in V. We illustrate the sequence R with examples from three different classes of grammars for which m is bounded. For each of the examples below let $T = \{a, b\}$, $n = 2$, and let V have the order $\langle a, b, X, Y\rangle$. (1) Let Γ be the class of regular grammars. Then $R = \langle a, b, aX, aY, bX, bY\rangle$. (2) Let Γ be the class of Chomsky normal form grammars. Then $R = \langle a, b, XX, XY, YX, YY\rangle$. (3) Let Γ be the class of restricted operator grammars. Then $R = \langle a, b, X, Y, aX, aY, bX, bY, Xa, Xb, Ya, Yb, aXa, aXb, aYa, aYb, bXa, bXb, bYa, bYb, XaX, XaY, XbX, XbY, YaX, YaY, YbX, YbY\rangle$.

For classes of grammars in which m is unbounded, the sequence R also depends on m. Two examples follow. (1) Let $T = \{a, b\}$, $n = 2$, $m = 2$, and let V have the order $\langle a, b, X, Y\rangle$. Let Γ be the class of all context free grammars. Then $R = \langle a, b, X, Y, aa, ab, aX, aY, ba, bb, bX, bY, Xa, Xb, XX, XY, Ya, Yb, YX, YY\rangle$. (2) Let $T = \{a\}$, $n = 3$, $m = 3$, and let V have the order $\langle a, X, Y, Z\rangle$. Let Γ be the class of Greibach normal form grammars. Then $R = \langle a, aX, aY, aZ, aXX, aXY, aXZ, aYX, aYY, aYZ, aZX, aZY, aZZ\rangle$.

We pause now for two observations. First, in every case the sequence R is independent of the number of productions p in the grammar. Second, once R is defined, the grammar enumerator that we describe below does not depend on the specific class of grammars. To change the enumerator from one class of grammars to another it is only necessary to replace the component that constructs the sequence R. Since this component is usually very simple, the grammar enumerator is only trivially dependent on the specific class of grammars being enumerated.

The operation of the grammar-enumerating schema, which enumerates all grammars in a structure class $\Gamma_{c,S}$, is straightforward. Each grammar G in $\Gamma_{c,S}$ is specified by a partially completed array A of positive integers with n rows, where row i has length S_i, for $1 \leqslant i \leqslant n$. Therefore the vector S represents the structure of the grammar G and the structure of the associated array A. Each element in A is an index into the sequence R.

EXAMPLE 6. Let $T = \{a, b, c\}$, $n = 2$, and $p = 4$. Then $V = \langle a, b, c, X, Y\rangle$. If Γ is the class of Chomsky normal form grammars then $R = \langle a, b, c, XX, XY, YX, YY\rangle$. Then for $S = \langle 3, 1\rangle$, the array

$$A = \begin{bmatrix} 1 & 2 & 5 \\ 3 & & \end{bmatrix} \text{ denotes the grammar } G\colon \begin{array}{l} X \to a \mid b \mid XY, \\ Y \to c. \end{array}$$

To enumerate $\Gamma_{c,S}$ it is only necessary to vary the array A sequentially over all possible values, where each element of A takes on all values from 1 to r, the cardinality of R. To describe this process we consider the elements of A to form a sequence as well as an array. The sequential order is defined as follows: array element A_{ij} precedes A_{ki} if $i < k$ and A_{ij} precedes A_{ik} if $j < k$.

The array A is *initialized* by setting all its elements to 1 and an element in A is *incremented* by increasing its value by 1. The process by which A successively takes on all its values requires the use of a *pointer*, a pair of *array indices* (i, j). First, A is initialized. The role of the pointer may be defined by the following three rules: (1) The pointer always points to the sequentially last element in A which can be incremented. (2) The element being pointed to is incremented. (3) That part of the array which follows the element being incremented is reinitialized. This process continues until no element exists which can be incremented. At this time A has taken on all possible values and the enumeration of $\Gamma_{c,S}$ is complete.

EXAMPLE 7. Let Γ be the class of Chomsky normal form grammars, let $T = \{a, b\}$, $n = 3$, $p = 5$, and $S = \langle 3, 1, 1\rangle$. Then $V = \langle a, b, X, Y, Z\rangle$ and $R = \langle a, b, XX, XY, XZ, YX, YY, YZ, ZX, ZY, ZZ\rangle$. A is initialized to

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & & \\ 1 & & \end{bmatrix}, \text{ which denotes the grammar } G_1\colon \begin{aligned} X &\to a \mid a \mid a, \\ Y &\to a, \\ Z &\to a. \end{aligned}$$

The pointer is set to the (3, 1) element in A and this element is incremented in steps of 1 to its maximum value of 11. Then

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & & \\ 11 & & \end{bmatrix}, \text{ which denotes the grammar } G_2\colon \begin{aligned} X &\to a \mid a \mid a, \\ Y &\to a, \\ Z &\to ZZ. \end{aligned}$$

The pointer is now set to the (2, 1) element, which is the last element that can be incremented. This element is incremented to 2, the (3, 1) element is reinitialized to 1, and the pointer returns to the (3, 1) element. Then

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & & \\ 1 & & \end{bmatrix}, \text{ which denotes the grammar } G_3\colon \begin{aligned} X &\to a \mid a \mid a, \\ Y &\to b, \\ Z &\to a. \end{aligned}$$

This process continues until

$$A = \begin{bmatrix} 11 & 11 & 11 \\ 11 & & \\ 11 & & \end{bmatrix}, \text{ which denotes the grammar } G_4\colon \begin{aligned} X &\to ZZ \mid ZZ \mid ZZ, \\ Y &\to ZZ, \\ Z &\to ZZ. \end{aligned}$$

At this time the enumeration of the structure class is complete.

4. AN IMPROVED GRAMMAR ENUMERATOR

It is apparent from the examples in the previous section that the grammar enumerator we have described has some weaknesses. In this section we consider

the following three: (1) It enumerates identical grammars which have their productions in different orders (e.g., $G_1 : X \to a \mid aX$ and $G_2 : X \to aX \mid a$). (2) It enumerates grammars which have identical productions (e.g., $G : X \to a \mid a \mid aX$). (3) It enumerates grammars which have nonterminals which do not appear on the left-hand side of any productions (e.g., G: $X \to a \mid aY$). These difficulties can be resolved by placing some simple restrictions on the enumerator. For problems (1) and (2) we require that all right-hand sides of productions with the same left-hand side be in the same order as the elements in R. This implies the following *order condition* on the elements in A: $A_{ij} < A_{ik}$ when $j < k$, for $1 \leqslant i \leqslant n$. However, it is impossible to satisfy this condition if for some S_i, where $1 \leqslant i \leqslant n$, $S_i > r$, where r is the cardinality of R. Any structure class with this property is omitted from the enumeration. For example, let Γ be the class of regular grammars, let $T = \{a\}$, $p = 5$, $n = 2$, and $S = \langle 4, 1\rangle$. Then $V = \langle a, X, Y\rangle$ and $R = \langle a, aX, aY\rangle$. Since $S_1 > r$, there cannot exist any grammars in the structure class $\Gamma_{c,S}$ satisfying this order condition. In problem (3) the enumeration of grammars having nonterminals which generate no productions occurs precisely when the structure S contains one or more entries with the value zero. This problem is eliminated by requiring each entry in S to be positive. However, this requirement cannot be satisfied for those complexity classes for which $p < n$, and therefore all such complexity classes are omitted from the enumeration.

The grammar-generating schema described above may readily be modified to conform to the order condition on the elements of A. When the array A is initialized, each element is set to the lowest possible value compatible with the order condition, and when an element is incremented, it may not be increased beyond the highest value compatible with the order condition. This process is illustrated in Example 8, which should be compared with Example 7.

EXAMPLE 8. Let Γ be the class of Chomsky normal form grammars, let $T = \{a, b\}$, $n = 3$, $p = 5$, and $S = \langle 3, 1, 1\rangle$. Then $V = \langle a, b, X, Y, Z\rangle$ and $R = \langle a, b, XX, XY, XZ, YX, YY, YZ, ZX, ZY, ZZ\rangle$. A is initialized to

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 1 & & \\ 1 & & \end{bmatrix}, \text{ which denotes the grammar } G_1\colon \begin{array}{l} X \to a \mid b \mid XX, \\ Y \to a, \\ Z \to a. \end{array}$$

The pointer is set to the (3, 1) element in A and this element is incremented in steps of 1 to its maximum value of 11. Then

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 1 & & \\ 11 & & \end{bmatrix}, \text{ which denotes the grammar } G_2\colon \begin{array}{l} X \to a \mid b \mid XX \\ Y \to a, \\ Z \to ZZ. \end{array}$$

The pointer is now set to the (2, 1) element, which is the last element that can be

incremented. This element is incremented to 2, the (3, 1) element is reinitialized to 1, and the pointer returns to the (3, 1) element. Then

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & & \\ 1 & & \end{bmatrix}, \text{ which denotes the grammar } G_3\colon \begin{array}{l} X \to a \mid b \mid XX, \\ Y \to b, \\ Z \to a. \end{array}$$

This process continues until

$$A = \begin{bmatrix} 9 & 10 & 11 \\ 11 & & \\ 11 & & \end{bmatrix}, \text{ which denotes the grammar } G_4\colon \begin{array}{l} X \to ZX \mid ZY \mid ZZ, \\ Y \to ZZ, \\ Z \to ZZ. \end{array}$$

At this time the enumeration of the structure class is complete.

This new grammar enumerator determines a new numbering scheme for a class of grammars Γ. However, it produces grammars in the same order as the original enumerator. Unlike the first version, the new enumerator is not exhaustive, since it omits many undesirable grammars, but it also does not repeat any grammars.

5. An Efficient Grammar Enumerator

Once a particular grammar has been enumerated some test must be applied to it to determine whether or not it is an acceptable solution to the specific inference problem under consideration. There are a number of reasons for rejecting a grammar produced by the improved enumerator. In this section we list 10 reasons for rejecting a grammar, each reason being formulated as a specific test, and we then show how information gained from the tests is used to significantly improve the efficiency of the grammar enumerator.

Test 1: *Complete equivalence.* Any two grammars are completely equivalent if one can be transformed into the other by an interchange of their nonterminal symbols. One member of each equivalence class is defined to be in the *canonical form* of the class. Any grammar not in the canonical form of its equivalence class of completely equivalent grammars is rejected by this test. Before describing the test, the canonical form will be characterized in terms of the sequence R of possible right-hand sides of productions. Let $[G]$ be the equivalence class of all grammars completely equivalent to a grammar G and let G_1 and G_2 be any two members of $[G]$. Let (i, j) be a pointer to, or the pair of array indices of, the sequentially first production in which G_1 differs from G_2, and let x and y be the righthand sides of these two productions. If x precedes y in R, then G_1 precedes G_2 in $[G]$. Since $[G]$ can be strictly ordered, the canonical form of $[G]$ is defined to be the sequentially first grammar in the sequence.

We note that this definition of the canonical form depends on the order of the sequence R, which in turn depends on the order of the sequence V.

In the actual test which is used, instead of examining an entire class $[G]$, only the individual grammar G is examined to determine whether or not it is in the canonical form. Let x and y be any two strings in R. Then x is *replaceable* by y if and only if x can be transformed into y by some interchange among the nonterminals. For example, let $V = \langle a, b, X, Y, Z\rangle$. Then aXY is not replaceable by aX, XaX is not replaceable by XaY, but $aXYbY$ is replaceable by $aYZbZ$. The test consists of a series of steps, in each of which one element of the array A associated with the grammar G and its corresponding string in R are examined. The steps are performed in the sequential order of A. At the beginning of each step a subset of the nonterminals is said to be *fixed* and the remainder is said to be *free*. Let x and y be any two strings in R. Then x is *freely replaceable* by y if and only if x can be transformed into y by some interchange among the free nonterminals. For example, let $V = \langle a, b, X, Y, Z\rangle$ and let X be fixed and Y and Z be free. Then $aXYbY$ is not freely replaceable by $aYZbZ$, but it is freely replaceable by $aXZbZ$. Initially, the initial nonterminal is fixed and all others are free. In each step one array element A_{ij} and its corresponding string in R, $x = R_{A_{ij}}$, are examined. If there exists a string y preceding x in R such that x is freely replaceable by y then G is not in canonical form. If no such y exists, all free nonterminals in x are changed to fixed and the next element in A is examined.

There are two ways in which a grammar can pass this test: (i) when at least $n - 1$ nonterminals are fixed, or (ii) when the entire array has been examined and there are two or more remaining free nonterminals. For case (i) it is only necessary to fix $n - 1$ nonterminals, since free replaceability requires at least two free nonterminals. For case (ii), grammars of this type may be considered "pathological" since this condition can only occur in disconnected grammars (see test 2). Except in the "pathological" case, for any equivalence class $[G]$ of completely equivalent grammars there is precisely one grammar in $[G]$ which can pass this test. This grammar is in the canonical form of the class. In the "pathological" case grammars which are not in canonical form can pass this test. This does not lead to any difficulties, since any such grammar will be rejected by the test for disconnected grammars.

EXAMPLE 9. Let Γ be the class of Chomsky normal form grammars, let $V = \langle a, b, c, d, X, Y, Z, W\rangle$, and $R = \langle a, b, c, d, XX, XY, XZ, XW, YX, YY, YZ, YW, ZX, ZY, ZZ, ZW, WX, WY, WZ, WW\rangle$.

$$
\begin{aligned}
G_1 :\ & X \to a \mid XY, \\
& Y \to b \mid YZ, \\
& Z \to c \mid ZW, \\
& W \to d;
\end{aligned}
\qquad\qquad
\begin{aligned}
G_2 :\ & X \to a \mid XY, \\
& Y \to b \mid YW, \\
& Z \to d, \\
& W \to c \mid WZ.
\end{aligned}
$$

For grammar G_1, initially X is fixed. At production (1, 2) Y is fixed, since no string in R preceding XY can freely replace XY. Similarly, at production (2, 2) Z is fixed. Since three of the four nonterminals are now fixed G_1 passes the test. For grammar G_2, initially X is fixed. At production (1, 2) Y is fixed. However, at production (2, 2) YW can be freely replaced by YZ, which precedes YW in R, and therefore G_2 fails the test.

Test 2: *Disconnected grammars.* A grammar is *connected* if each nonterminal symbol occurs in some string which can be derived from the initial nonterminal; otherwise it is *disconnected.* Thus in Example 10 G_1 is connected, but G_2 is disconnected, since X does not derive any strings containing Z.

EXAMPLE 10.

$$G_1 : X \to a \mid YY, \quad Y \to b \mid XY; \qquad G_2 : X \to a \mid YY, \quad Y \to b \mid XY, \quad Z \to c \mid YZ.$$

Test 3: *Blocking grammars.* A grammar is *nonblocking* if each nonterminal symbol generates a terminal string; otherwise it is *blocking.* Thus in Example 11 G_1 is nonblocking, but G_2 is blocking because Y generates no terminal strings.

EXAMPLE 11.

$$G_1 : X \to a \mid XX; \qquad G_2 : X \to a \mid XX \mid XY, \quad Y \to YY.$$

Test 4: *Merging nonterminals.* Any set of two or more nonterminals from a grammar may be *merged* if their productions are identical or differ only by nonterminals which are members of this set. A grammar fails this test if it has such a set, called a *merge set.* Thus in Example 12 G_1 passes the test since no merge set exists. G_2 fails the test with the merge set $\{Y, Z\}$, since Y and Z generate identical productions. G_3 also fails with the merge set $\{Y, Z\}$, since Y and Z differ only in the productions $Y \to eZ$ and $Z \to eY$. For G_4 none of the possible pairs $\{Y, Z\}$, $\{Y, W\}$, or $\{Z, W\}$ may be merged, but the triple $\{Y, Z, W\}$ is a merge set, and so G_4 fails the test.

EXAMPLE 12.

$$G_1 : X \to a \mid bY \mid cY, \quad Y \to dX \mid eY; \qquad G_2 : X \to a \mid bY \mid cZ, \quad Y \to dX \mid eZ, \quad Z \to dX \mid eZ;$$

$$G_3 : X \to a \mid bY \mid cZ, \quad Y \to dX \mid eZ, \quad Z \to dX \mid eY; \qquad G_4 : X \to a \mid bY \mid cZ, \quad Y \to dX \mid eZ, \quad Z \to dX \mid eW, \quad W \to dX \mid eY.$$

Test 5: *Direct substitution.* The productions generated by one nonterminal can be *directly substituted* into the productions generated by the remaining nonterminals wherever the nonterminal being replaced generates just one production and the right-hand side of this production consists of a single occurrence of another nonterminal. A grammar fails this test if direct substitution is possible. Thus in Example 13 G_1 passes the test, but G_2 fails, since X can be directly substituted for Y.

EXAMPLE 13.

$$G_1 : X \to a \mid bX; \qquad G_2 : X \to Y, \quad Y \to a \mid bY.$$

Since *ambiguous grammars* (Hopcroft and Ullman, 1969) are considered undesirable solutions to some inference problems, we include tests for ambiguity. Although the *ambiguity problem* for context-free grammars is, in general, undecidable (Hopcroft and Ullman, 1969), there are some types of ambiguity which can be detected by general tests.

Test 6: *Ambiguity—circular nonterminals.* A connected, nonblocking grammar is ambiguous if it contains a set of one or more nonterminals $\{Y, Z,...\}$ such that $Y \Rightarrow Z \Rightarrow \cdots \Rightarrow Y$. Thus in Example 14 G_1 fails the test since the string a has the derivation $X \Rightarrow X \Rightarrow a$. G_2 fails since the string a has the derivation $X \Rightarrow Y \Rightarrow X \Rightarrow a$.

EXAMPLE 14.

$$G_1 : X \to a \mid X \mid bY, \quad Y \to c \mid dX; \qquad G_2 : X \to a \mid Y \mid bX, \quad Y \to c \mid X \mid dY.$$

Test 7: *Ambiguity—left and right recursion.* A connected, nonblocking grammar is ambiguous if it is both *left* and *right recursive* in some nonterminal symbol. That is, there exists a nonterminal Y and strings α and β such that $Y \Rightarrow^+ \alpha Y$ and $Y \Rightarrow^+ Y\beta$. This follows from the derivations $Y \Rightarrow^+ \alpha Y \Rightarrow^+ \alpha Y\beta$ and $Y \Rightarrow^+ Y\beta \Rightarrow^+ \alpha Y\beta$. Thus in Example 15 each of the three grammars is both left and right recursive and therefore ambiguous. G_1 has the derivation $X \Rightarrow XcX$. G_2 has the derivations $X \Rightarrow bY \Rightarrow beX$ and $X \Rightarrow Xc$. G_3 has the derivations $Y \Rightarrow X \Rightarrow bY$ and $Y \Rightarrow Yd$.

EXAMPLE 15.

$$G_1 : X \to a \mid bX \mid XcX; \qquad G_2 : X \to a \mid bY \mid Xc, \quad Y \to d \mid eX \mid fY;$$

$$G_3 : X \to a \mid bY, \quad Y \to c \mid X \mid Yd.$$

Test 8: *Missing terminal symbols.* By any common definition of compatibility, if a grammar is missing one or more of the terminal symbols present in the sample it cannot be compatible with the sample. Thus in Example 16 G is incompatible with the sample S^+, since G does not contain the terminal symbol c.

EXAMPLE 16.

$$S^+ = \{a,\ ab\ \ abc\}; \qquad G : X \to a \mid bY,$$
$$Y \to aX.$$

While the eight tests described above require only the direct observation of the grammar, the tests for ambiguity and for incompatibility with the sample described below require the enumeration of the language of the grammar in order of increasing length. It is well known that this can readily be done for any context free grammar. In particular, a *left-most enumeration* is used. That is, for any mixed string $\alpha = xN_i\beta$ which appears in the enumeration, where $x \in T^*$, $N_i \in N$, and $\beta \in V^*$, if N_i generates the productions $N_i \to \gamma_{i1} \mid \gamma_{i2} \mid \cdots \mid \gamma_{iS_i}$, the string α is replaced by the set of strings $\{x\gamma_{i1}\beta,\ x\gamma_{i2}\beta,\ldots,\ x\gamma_{iS_i}\beta\}$.

Test 9: *Ambiguity.* When a new string is produced a search is made among all strings of the same length to determine whether or not the string has previously been produced. If this has occurred the grammar is ambiguous. Thus the grammar in Example 17 is ambiguous since $X \Rightarrow YY \Rightarrow aY \Rightarrow aaY$ and $X \Rightarrow YY \Rightarrow aYY \Rightarrow aaY$. This ambiguity is not detected by either of the two previous ambiguity tests, test 6 and test 7.

EXAMPLE 17.

$$G : X \to YY,$$
$$Y \to a \mid aY.$$

Test 10: *Incompatibility with the sample.* As previously noted, a variety of definitions can be given for the compatibility of a grammar with a sample. In each case incompatibility can be detected by enumerating the language produced by the grammar and comparing it to the sample.

EXAMPLE 18. $S^+ = \{a, b, ab\}$; $G : X \to a \mid bX$.

For this example we assume criterion 1 for compatibility given above, which requires that $S^+ \subset L(G)$. G is incompatible with S^+, since $b \in S^+$ and $b \notin L(G)$.

If a grammar fails a particular test, then its *failure point* with respect to that test is the sequentially first production at which the failure of the test can be definitely established. The significance of this definition derives from the fact that as long as the productions in the grammar up to and including the production at the failure point remain unchanged, the grammar must fail the test no matter

what changes occur in the productions following the failure point. Below we describe procedures for determining the failure points for test 1 and test 10. The interested reader may apply the same principle to construct procedures to determine the failure points for the remaining tests.

Test 1: *Complete equivalence—failure point.* The failure point of a grammar which fails this test is the sequentially first production for which the string x on the right-hand side of the production can be freely replaced by a string y which precedes x in R. Thus the grammar in Example 19 has the failure point (1, 2) for this test, since XZ can be freely replaced by XY and since XY precedes XZ in R.

EXAMPLE 19. Let Γ be the class of Chomsky normal form grammars, let $V = \langle a, b, c, X, Y, Z\rangle$, and $R = \langle a, b, c, XX, XY, XZ, YX, YY, YZ, ZX, ZY, ZZ\rangle$:

$$G: X \to a \mid XZ,$$
$$Y \to b,$$
$$Z \to XY \mid YZ.$$

Test 10: *Incompatibility with the sample—failure point.* Criterion 2 for compatibility, given above, is appropriate to demonstrate the failure point for this test. With this definition of compatibility it is necessary that $S^+ \subset L(G)$ and $S^- \subset \overline{L(G)}$. Let $L^i(G)$ be the subset of $L(G)$ whose strings have length i, let $\overline{L^i(G)}$ be defined to be $T^i - L^i(G)$, and let $S^{+,i}$ and $S^{-,i}$ be the corresponding subsets of S^+ and S^-. Assume that for some $k \geqslant 1$ and for all $i < k$, $S^{+,i} \subset L^i(G)$ and $S^{-,i} \subset \overline{L^i(G)}$, but $S^{+,k} \not\subset L^k(G)$ or $S^{-,k} \not\subset \overline{L^k(G)}$. We define two *failure sets* of strings F^+ and F^- as follows: $F^+ = S^{+,k} - L^k(G)$ and $F^- = S^{-,k} - \overline{L^k(G)} = S^{-,k} \cap L^k(G)$. It is the set of strings $F = F^+ \cup F^-$ that "causes" the grammar to fail the test. Since each string $x \in L(G)$ is produced by a subset (possibly all) of the productions in G, it is possible to identify for any x the sequentially last production in this subset. For any subset $L' \subset L(G)$ the sequentially last production used to produce L' is the last production in the set of last productions for all $x \in L'$. A failure point is defined for each string in F in the following manner. The failure point of any string $y \in F^-$ is at the sequentially last production in the subset of productions in G which produce y. The failure point of any string $z \in F^+$ is at the sequentially last production in the subset of productions in G which produce the strings in $L(G)$ whose length does not exceed $k = l(z)$. The failure point of G for this test is the first of the failure points of all the strings in F.

EXAMPLE 20. G: $X \to a \mid aX \mid bX \mid cX$; $S^+ = \{a, aa, ab, ac\}$, $S^- = \{b, c, ba, ca\}$.

Table I below shows the enumeration of $L(G)$ up to length 2.

TABLE I

Enumeration of $L(G)$ up to Length 2

Length = 1		Length = 2	
String	Last production	String	Last Production
X	(0, 0)	aX	(1, 2)
a	(1, 1)	bX	(1, 3)
		cX	(1, 4)
		aa	(1, 2)
		ba	(1, 3)
		ca	(1, 4)

For strings of length 1, $S^{+,1} \subset L^1(G)$ and $S^{-,1} \subset \overline{L^1(G)}$. For strings of length 2, $S^{+,2} = \{aa, ab, ac\}$, $S^{-,2} = \{ba, ca\}$, $L^2(G) = \{aa, ba, ca\}$, $\overline{L^2(G)} = \{ab, ac, bb, bc, cb, cc\}$, $F^+ = \{ab, ac\}$, and $F^- = \{ba, ca\}$. The failure points of the strings ab, ac, ba, and ca are, respectively, (1, 4), (1, 4), (1, 3), and (1, 4). Therefore the failure point of G for this test is (1, 3).

For each grammar which fails any of the tests an (*overall*) *failure point* is determined in the following way. First, the direct observation tests are applied to the grammar. If it fails one or more of the tests its overall failure point is the first of its failure points for the tests which it fails. No further tests are applied. If the grammar passes all eight direct observation tests then language enumeration begins. If an ambiguity is detected the failure point for the grammar is its failure point for this test. If no ambiguity is detected the test for compatibility is applied. If the grammar fails this test the failure point for the grammar is its failure point for this test. Any grammar which passes the last test is a valid solution to the inference problem. The grammar in Example 21 fails test 1 (complete equivalence) at the failure point (1, 2) and test 2 (disconnection) at the failure point (3, 1). Therefore its overall failure point is (1, 2).

EXAMPLE 21.

$$G: X \to a \mid bZ,$$
$$Y \to cX \mid dZ,$$
$$Z \to aX.$$

When the failure point of a grammar G is determined, the testing process is complete for this grammar and the grammar enumeration process begins again. However, the element in the array A corresponding to G which is incremented is not the last possible one, but the element identified by the failure point of G. This conceptually simple mechanism omits the vast majority of invalid grammars from the enumeration.

EXAMPLE 22. Let Γ be the class of regular grammars and let $V = \langle a, b, c, d, X, Y, Z\rangle$ and $R = \langle a, b, c, d, aX, aY, aZ, bX, bY, bZ, cX, cY, cZ, dX, dY, dZ\rangle$. When A has the value

$$A = \begin{bmatrix} 1 & 10 \\ 11 & 16 \\ 5 & \end{bmatrix},$$ which corresponds to the grammar G_1: $X \to a \mid bZ,$
$Y \to cX \mid dZ,$
$Z \to aX,$

its failure point, determined in Example 21 above, is (1, 2). Therefore A is incremented at this element and becomes

$$A = \begin{bmatrix} 1 & 11 \\ 1 & 2 \\ 1 & \end{bmatrix},$$ which corresponds to the grammar G_2: $X \to a \mid cX,$
$Y \to a \mid b,$
$Z \to a.$

All grammars omitted from the enumeration at this time would have had the failure point (1, 2).

One further technique to improve the overall efficiency is described here. If a grammar G passes a particular test, then the *success point* for that test is the sequentially first production at which the success of the test can be definitely established. As long as the productions in the grammar up to and including the production at the success point remain unchanged, the grammar must pass the test, no matter what changes occur in the productions following the success point. Therefore the test need not be applied to the sequence of grammars following G which satisfy this property. However, as soon as the production at the success point or any preceding production is modified, the test must again be applied. Below we describe a procedure for determining the success point for test 1 (complete equivalence). Once again the interested reader may apply the same principle to construct procedures which determine success points for the remaining tests.

Test 1: *Complete equivalence-success point.* The test for complete equivalence is described in terms of free and fixed nonterminals. It follows immediately from the description of the test that, except in the "pathological" case, the success point is the first production at which the number of free nonterminals is reduced to one or zero. In the "pathological" case the success point is the last production in the grammar. Thus in Example 23 the success point is (1, 2), since at this point the nonterminal symbols X and Y are fixed and only Z remains free.

EXAMPLE 23.

$$G: X \to aX \mid bY,$$
$$Y \to c \mid aZ,$$
$$Z \to dX.$$

This concludes the description of the third and last of our grammar enumerators. Although many details are provided, the essential concepts are few. They may be summarized as follows: (i) the use of a grammar-generating schema to enumerate the grammars; (ii) the application of tests to the grammars; (iii) the use of failure points to eliminate invalid grammars from the enumeration; and (iv) the use of success points to eliminate unnecessary test applications. In the next section we supply empirical evidence to validate our claims for the utility of the techniques used in these enumerators.

6. Empirical Results and Conclusions

The efficient grammar enumerator described above has been implemented as a computer program in a number of variations. The component that constructs the sequence R of possible right-hand sides of productions has been written in three variations to allow for the following three class of grammars: regular grammars, Chomsky normal form grammars, and restricted operator grammars. The component that tests for compatibility has also been written in three variations according to the three criteria for compatibility given in Section 2. In addition, the component that reads in the sample strings was written in two variations, the first for positive samples only, as required by compatibility criteria 1 and 3, and the second for both positive and negative samples, as required by criterion 2. All procedures described here have been implemented as computer programs in Fortran IV. The programs were compiled by the IBM System/360 Fortran H (optimizing) compiler and executed on an IBM System/370 Model 158 computer. A typical version of the efficient grammar enumerator consists of approximately 1600 Fortran statements.

The relative efficiency of the different enumerators can be observed in the four examples given below. At the same time, they illustrate the use of different criteria for the compatibility of a grammar with a sample and the use of different classes of grammars. In each case, the language to be identified is the language of arithmetic expressions with the operations of addition, subtraction, unary minus, and with parentheses. The sample of strings $S^+ = \{a, -a, a + a, a - a, (a), -a + a, -a - a, -(a), (-a)\}$ is used in each of the following examples. In Examples 24, 25, and 26 the class of restricted operator grammars over the terminal vocabulary $T = \{a, +, -, (,)\}$ was enumerated, and in Example 27 the class of Chomsky normal form grammars over the same terminal vocabulary was enumerated.

Example 24. Criterion 1 for compatibility, which requires that $S^+ \subset L(G)$, was employed. The restricted operator grammar $G_1 : X \to a \mid aX \mid +X \mid -X \mid (X)$, which has complexity $c = 11$, was produced in 7.84 sec.

EXAMPLE 25. Criterion 2 for compatibility, which requires that $S^+ \subset L(G)$ and $S^- \subset \overline{L(G)}$, was employed. The negative sample used was $S^- = \{aa, +a, ++a, -+a, a(a), aaaa\}$. The restricted operator grammar $G_2 : X \to a \mid + \mid - \mid -X \mid aXa \mid (X)$, which has complexity $c = 16$, was produced in 31.36 sec.

EXAMPLE 26. Criterion 3 for compatibility, which requires $L(G)$ to be identical to S^+ up to the length of the longest string in S^+, was employed. The restricted operator grammar

$$G_3 : X \to Y \mid -Y \mid X + Y \mid X - Y,$$
$$Y \to a \mid (X),$$

which has complexity $c = 23$, was produced in 17 min, 9.95 sec. We note that G_3 is precisely the grammar for the language of arithmetic expressions described above.

EXAMPLE 27. Criterion 1 for compatibility was employed. The Chomsky normal form grammar

$$G_4 : X \to a \mid) \mid YX,$$
$$Y \to a \mid + \mid - \mid (,$$

which has complexity $c = 30$, was produced in 4.98 sec.

A counter was inserted into the programs that produced these grammars in order to determine their sequence numbers in the enumerations. The sequence numbers are given in Table II. In addition, since the original enumerator and the improved enumerator are so simple and regular, it was possible to compute the sequence numbers of the above grammars for these two enumerators. For comparison purposes, these sequence numbers are also given in Table II.

TABLE II

Sequence Numbers of Enumerated Grammars

Grammar	Sequence number		
	Original enumerator	Improved enumerator	Efficient enumerator
$G_1 : X \to a \mid aX \mid +X \mid -X \mid (X)$	9,006,473	1,149,751	2,643
$G_2 : X \to a \mid + \mid - \mid -X \mid aXa \mid (X)$	701,675,530	56,693,036	9,827
$G_3 : X \to Y \mid -Y \mid X+Y \mid X-Y$ $Y \to a \mid (X)$	2,225,706,812,694	13,135,930,021	355,576
$G_4 : X \to a \mid) \mid YX$ $Y \to a \mid + \mid - \mid ($	45,430,329	861,276	1,140

The reduction in the number of grammars enumerated between the original enumerator and the improved version is the result of the simple modifications outlined in Section 4. The reduction in the number of grammars enumerated between this improved enumerator and the final version is entirely due to the use of failure points to bypass invalid grammars. Looking at grammar G_3 in Table II, we see that the first improvement reduces the sequence number by two orders of magnitude and the second improvement reduces the sequence number by a further five orders of magnitude. It is this significant increase in efficiency that makes it at all reasonable to use enumerative methods for realistic inference problems.

RECEIVED: October 17, 1975; REVISED: May 28, 1976

REFERENCES

BIERMANN, A., AND FELDMAN, J. A. (1972), A survey of results in grammatical inference, *in* "Frontiers of Pattern Recognition" (S. Watanabe, Ed.), Academic Press, New York.

BLUM, M. (1967), On the size of machines, *Inform. Contr.* **11**, 257–265.

CRESPI-REGHIZZI, S. (1970), "The Mechanical Acquisition of Precedence Grammars," Report UCLA-ENG-7054, School of Engineering and Applied Science, University of California, 1970.

FELDMAN, J. A. (1967), "First Thoughts on Grammatical Inference," Artificial Intelligence Memorandum No. 55, Computer Science Department, Stanford University, August 1967.

FELDMAN, J. A. (1972), Some decidability results on grammatical inference and complexity, *Inform. Contr.* **20**, 244–262.

FELDMAN, J. A., GIPS, J., HORNING, J. J., AND REDER, S. (1969), "Grammatical Complexity and Inference," Technical Report No. CS125, Computer Science Department, Stanford University.

FU, K. S. (1974), "Syntactic Methods in Pattern Recognition," Academic Press, New York.

GOLD, E. M. (1967), Language identification in the limit, *Inform. Contr.* **10**, 447–474.

GOLD, E. M. (1973), Current approaches to the inference of phrase structure grammars, unpublished paper, Département d'Informatique, Université de Montréal.

HOPCROFT, J. E., AND ULLMAN, J. D. (1969), "Formal Languages and their Relation to Automata," Addison–Wesley, Reading, Mass.

HORNING, J. J. (1969), "A Study of Grammatical Inference," Technical Report No. CS139, Computer Science Department, Stanford University.

PAO, T. W. L. (1969), "A Solution to the Syntactic Induction–Inference Problem for a Non-Trivial Subset of Context-Free Languages," Report No. 79-19, The Moore School of Electrical Engineering, University of Pennsylvania.

SOLOMONOFF, R. (1959), A new method for discovering the grammars of phrase structure languages, *Inform. Process.* 258–290.

SOLOMONOFF, R. (1964), A formal theory of inductive inference, *Inform. Contr.* **7**, 1–22, 224–254.

WHARTON, R. M. (1974), Approximate language identification, *Inform. Contr.* **26**, 236–255.

Printed in Belgium